The DUNE Buildsystem HOWTO

Christian Engwer*

August 2, 2007

*Interdisziplinäres Zentrum für Wissenschaftliches Rechnen, Universität Heidelberg,
Im Neuenheimer Feld 368, D-69120 Heidelberg, Germany

http://www.dune-project.org/

Contents

1 Structure of DUNE **2**

2 Toolchain **3**
2.1 Autotools . . . 3
2.2 Makefile.am . . . 4
2.2.1 Overview . . . 4
2.2.2 Building Documentation . . . 6
2.3 configure.ac . . . 7
2.4 autogen.sh . . . 9
2.5 dunecontrol . . . 9

3 Creating a new Dune module **11**

4 Creating a new Dune application **12**

5 Futher documentation **14**

1 Structure of DUNE

DUNE consists of several independent modules:

- dune-common
- dune-grid
- dune-istl
- dune-disc
- dune-fem

These modules interact very tightly and depend on each other.
The build system is structured as follows:

- Modules are build using the GNU autotools.
- Each module has a set of modules it depends on, these modules have to be built before building the module itself.
- Each module has a file dune.module which hold dependecies and other information regarding the module.
- The modules can be built in the appropriate order using the dunecontrol script (shipped with dune-common)

The reasons to use the GNU autotools for DUNE were the following

- We need platform independent build.
- Enabling or disabling of certain features depending on features present on the system.
- Creations of libraries on all platforms.
- Easy creation of portable but flexible Makefiles.

The reasons to add the dunecontrol script and the dune.module description files were

- One tool to setup all modules (autotools can only work on one module).
- Automatic dependency tracking.
- Automatic collection of commandline parameters (configure needs special commandline parameters for all modules it uses)

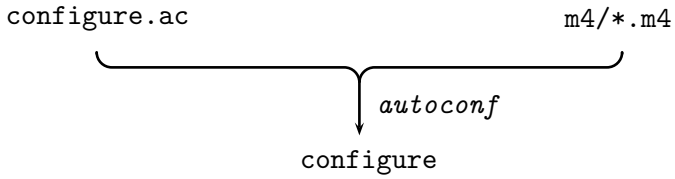
2 Toolchain

2.1 Autotools

Software is generally developed to be used on multiple platforms. Since each of these platforms have different compilers, different include files, there is a need to write Makefiles and build scripts so that they can work on a variety of platforms. The free software community (Project GNU), faced with this problem, devised a set of tools to generate Makefiles and build scripts that work on a variety of platforms. If you have downloaded and built any GNU software from source, you are familiar with the `configure` script. The `configure` script runs a series of tests to determine information about your machine.

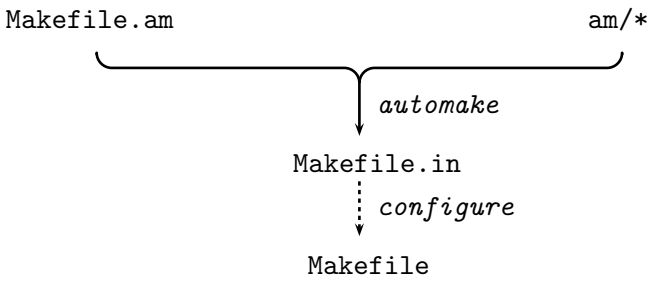
The autotools simplify the generation of portable Makefiles and configure scripts.

autoconf
`autoconf` is used to create the `configure` script. `configure` is created from `configure.ac`, using a set of `m4` files.



How to write a `configure.ac` for `DUNE` is described in 2.3

automake
`automake` is used to create the `Makefile.in` files (needed for `configure`) from `Makefile.am` files and, using a set of include files located in the `am` directory. These include files provide additional features, not provided by the standard `automake` (see 2.2.2). The `am` directory is in the `dune-common` module and each module intending to use one of these includes has to create symlink; this is usually done by `autogen.sh` (see 2.4).



Information on writing a `Makefile.am` is described in 2.2

libtool
`libtool` is a wrapper around the compiler and linker. It offers a generic interface for creating static and shared libraries, regardless of the platform it is running on.

`libtool` hides all the platform specific aspects of library creation and library usage. When linking a library or an executable you (or `automake`) can call the compiler via `libtool`. `libtool` will then take care of

- *platform specific* commandline parameters for the linker
- library dependencies

configure

`configure` will run the set of tests specified in your `configure.ac`. Using the results of these tests configure can check that all necessary features (libs, programs, etc.) are present and can activate and deactivate certain features of the module depending on the feature of the system.

For example `configure` in `dune-grid` will search for the ALUGrid library and depending on the result enable or disable `Dune::ALU3dGrid`.

Many test will also store their results in the `config.h` header file. A headerfile can then use an `#ifdef` statement to disable parts of the code that don't work without a certain feature. This can be used in the applications aswell as in the headers in a `DUNE` module. When we stick to the example of the ALUGrid library `config.h` will contain a `#define HAVE_ALUGRID` if ALUGrid was found.

The `config.h` file is created by configure from a `config.h.in` file, which is automatically created from the list of tests used in the `configure.ac`.

2.2 Makefile.am

2.2.1 Overview

Let's start off with a simple program *hello* built from `hello.c`. As automake is designed to build and install a package it needs to know

- what programs it should build
- where to put them when installing
- which sources to use

The `Makefile.am` thus looks like this:

```
noinst_PROGRAMS = hello
hello_SOURCES = hello.c
```

This would build hello and won't install it when `make install` is called. Using `bin_PROGRAMS` instead of `noinst_PROGRAMS` would install the hello-binary into a *`prefix`*`/bin` directory which we don't want to do with most of the `DUNE` applications.

Building more programs with a couple of source-files works like this

```
noinst_PROGRAMS = hello bye

hello_SOURCES = common.c common.h hello.c
bye_SOURCES = common.c common.h bye.c parser.y lexer.l
```

automake has more integrated rules than the standard make, the example above would automatically use yacc/lex to create `parser.c/lexer.c` and build them into the *bye* binary.

Make-Variables may be defined and used as usual:

```
noinst_PROGRAMS = hello bye

COMMON = common.c common.h

hello_SOURCES = $(COMMON) hello.c
bye_SOURCES = $(COMMON) bye.c parser.y lexer.l
```

Even normal make-rules may be used in a `Makefile.am`.

Using flags

Compiler/linker/preprocessor-flags can be set either globally:

```
noinst_PROGRAMS = hello bye

AM_CPPFLAGS = -DDEBUG

hello_SOURCES = hello.c
bye_SOURCES = bye.c
```

or locally:

```
noinst_PROGRAMS = hello bye

hello_SOURCES = hello.c
hello_CPPFLAGS = -DHELLO

bye_SOURCES = bye.c
bye_CPPFLAGS = -DBYE
```

The local setting overrides the global one, thus

```
hello_CPPFLAGS = $(AM_CPPFLAGS) -Dmyflags
```

may be a good idea.

It is even possible to compile the same sources with different flags:

```
noinst_PROGRAMS = hello bye

hello_SOURCES = generic-greeting.c
hello_CPPFLAGS = -DHELLO

bye_SOURCES = generic-greeting.c
bye_CPPFLAGS = -DBYE
```

Perhaps you're wondering why the above examples used `AM_CPPFLAGS` instead of the normal `CPPFLAGS`? The reason for this is that the variables `CFLAGS`, `CPPFLAGS`, `CXXFLAGS` etc. are considered *user variables* which may be set on the commandline:

```
make CXXFLAGS="-O2000"
```

This would override any settings in Makefile.am which might be necessary to build. Thus, if the variables should be set even if the user wishes to modify the values, you should use the `AM_*` version.

The real compile-command always uses both `AM_`*`VAR`* and *`VAR`*. Options that autoconf finds are stored in the user variables (so that they may be overridden)

Commonly used variables are:

- `AM_CPPFLAGS`: flags for the C-Preprocessor. This includes preprocessor defines like `-DNDEBUG` and include pathes like `-I/usr/local/package/include`
- `AM_CFLAGS`, `AM_CXXFLAGS`: flags for the compiler (-g, -O, ...). One difference between these and the `CPPFLAGS` is that the linker will get `CFLAGS`/`CXXFLAGS` and `LDFLAGS` but not `CPPFLAGS`

- AM_LDFLAGS options for the linker
- LDADD: libraries to link to a binary
- LIBADD: libraries to add to a library
- SOURCES: list of source-files (may include headers as well)

Conditional builds

Some parts of DUNE only make sense if certain addon-packages were found. autoconf therefore defines *conditionals* which automake can use:

```
if OPENGL
  PROGS = hello glhello
else
  PROGS = hello
endif

hello_SOURCES = hello.c

glhello_SOURCES = glhello.c hello.c
```

This will only build the *glhello* program if OpenGL was found. An important feature of these conditionals is that they work with any make program, even those without a native *if* construct like GNU-make.

Default targets

An automake-generated Makefile does not only know the usual *all*, *clean* and *install* targets but also

- **tags** travel recursively through the directories and create TAGS-files which can be used in many editors to quickly find where symbols/functions are defined (use emacs-format)
- **ctags** the same as "tags" but uses the vi-format for the tags-files
- **dist** create a distribution tarball
- **distcheck** create a tarball and do a test-build if it really works

2.2.2 Building Documentation

If you want to build documentation you might need additional make rules. DUNE offers a set of predefined rules to create certain kinds of documentation. Therefor you have to include the appropriate rules from the am/ directory. These rules are stored in the dune-common/am/ directory. If you want to use these any of these rules in your DUNE module or application you will have to create a symbolic link to dune-common/am/. The creation of this link should be done by the autogen.sh script.

html pages

Webpages are created from wml sources, using the program wml (http://thewml.org/).
$(top_srcdir)/am/webstuff containes the necessary rules.

Listing 1 (File Makefile.am)

```
# $Id: Makefile.am 4961 2007-07-30 20:03:01Z mblatt $

# also build these sub directories
SUBDIRS = devel doxygen layout buildsystem

# only build html pages, if documentation is enabled
if BUILD_DOCS
 PAGES = view-concept.html installation-notes.html contrib-software.html
endif

# automatically create these web pages
all: $(PAGES)

# setting like in dune-web
CURDIR=doc
# position of the web base directory,
# relative to $(CURDIR)
BASEDIR=..
EXTRAINSTALL=example.opts

# install the html pages
docdir=$(datadir)/doc/dune-common
doc_DATA = $(PAGES) example.opts
EXTRA_DIST = $(PAGES) example.opts

# include rules for wml -> html transformation
include $(top_srcdir)/am/webstuff

# remove html pages on ``make clean''
SVNCLEANFILES = $(PAGES)
clean-local:
    if test -e $(top_srcdir)/doc/doxygen/Doxydep; then rm -rf $(SVNCLEANFILES); fi

# include further rules needed by Dune
include $(top_srcdir)/am/global-rules
```

LaTeXdocuments

In order to compile LaTeXdocuments you can include `$(top_srcdir)/am/latex`. This way you get rules for creation of DVI files, PS files and PDF files.

SVG graphics

SVG graphics can be converted to png, in order to include them into the web page. This conversion can be done using inkscape (`http://www.inkscape.org/`). `$(top_srcdir)/am/inkscape.am` offers the necessary rules.

2.3 configure.ac

`configure.ac` is a normal text file that contains several `autoconf` macros. These macros are evaluated my the `m4` macro processor and transformed into a shell script.

Listing 2 (File dune-common/configure.ac)

```
#! /bin/bash
# $Id: configure.ac 4951 2007-06-27 19:31:55Z christi $
# Process this file with autoconf to produce a configure script.
AC_INIT(dune-common, 1.0beta5, dune@dune-project.org)
AM_INIT_AUTOMAKE
```

```
AC_CONFIG_SRCDIR([common/stdstreams.cc])
AM_CONFIG_HEADER([config.h])

# check all dune-module stuff
DUNE_CHECK_ALL_M

# preset variable to path such that #include <dune/...> works
AC_SUBST([DUNE_COMMON_ROOT], '$(top_srcdir)')
AC_SUBST([AM_CPPFLAGS], '-I$(top_srcdir)')
AC_SUBST([LOCAL_LIBS], '$(top_builddir)/common/libcommon.la')

DUNE_SUMMARY_ALL

echo
echo Note: Most of the libraries checked for above are only used for the self-test
echo of Dune.  The library itself will build and the headers will work even if
echo ALBERTA, MPI, etc. cannot be found.  An exception to this are UG and AmiraMesh
echo which need to be found right now if you want to use them later.
echo

# write output
AC_CONFIG_FILES([Makefile
     lib/Makefile
     bin/Makefile
     common/Makefile
     common/test/Makefile
     doc/Makefile
     doc/devel/Makefile
     doc/layout/Makefile
     doc/doxygen/Makefile
     doc/buildsystem/Makefile
     m4/Makefile
     am/Makefile
     bin/wmlwrap
     bin/check-log-store
     dune-common.pc])
AC_OUTPUT

chmod +x bin/wmlwrap
chmod +x bin/check-log-store
```

We offer a set of macros that can be used in your `configure.ac`:

- `DUNE_CHECK_ALL_M` runs all checks usually needed by a *DUNE module*. This macros takes list of other `DUNE` modules it should search for as parameters.

  ```
  DUNE_CHECK_ALL_M([dunecommon], [dunegrid])
  ```

  will search for `dune-common` and `dune-grid` (Attention: you have to provide the modules in such an order that the dependencies are checked already).

- `DUNE_CHECK_ALL` same as `DUNE_CHECK_ALL_M`, except that it only runs the tests needed for a *DUNE application*

- `DUNE_SUMMARY_ALL` prints information on the results of all major checks run by `DUNE_CHECK_ALL` or `DUNE_CHECK_ALL_M`.

`DUNE_CHECK_ALL` and `DUNE_CHECK_ALL_M` define certain variables that can be used in the `configure` script or in the `Makefile.am`:

- DUNE *MODULE* _CPPFLAGS
- DUNE *MODULE* _LDFLAGS
- DUNE *MODULE* _LIBS
- DUNE *MODULE* ROOT

The last step to a complete `configure.ac` is that you tell autoconf which files should be generated by `configure`. Therefor you add an `AC_CONFIG_FILES([`*WhiteSpaceSeparatedListOfFiles*`])` statement to your `configure.ac`. The list of files should be the list of files that are to be generated, not the input – i.e. you would write

```
AC_CONFIG_FILES([Makefile doc/Makefile])
```

end not

```
AC_CONFIG_FILES([Makefile.in doc/Makefile.in])
```

After you told `autoconf` which files to create you have to actually trigger their creation with command `AC_OUTPUT`

2.4 autogen.sh

The `autogen.sh` script is used to bring the freshly checked out module into that state that you expect from a module received via the tarball. That means it runs all necessary stepts so that you can call `configure` to setup your module. In the case of `DUNE` this means that `autogen.sh` runs

- `libtoolize` (prepare the module for `libtool`)
- `aclocal` (collect all `autoconf` macros needed for this module)
- `autoheader` (create the `config.h.in`)
- `automake` (create the `Makefile.in`)
- `autoconf` (create `configure`)

If needed it will also create the symbolic link to the `dune-common/am/` directory (see 2.2.2).

2.5 dunecontrol

`dunecontrol` helps you building the different `DUNE` modules in the appropriate order. Each module has a `dune.module` file which contains information on the module needed by `dunecontrol`.

`dunecontrol` searches for `dune.module` files recursively from where you are executing the program. For each `DUNE` module found it will execute a `dunecontrol` command. All commands offered by `dunecontrol` have a default implementation. This default implementation can be overwriten and extended in the `dune.module` file.

The commands you are interested in right now are

- `autogen` runs `autogen.sh` for each module. A list of directories containing `dune.module` files and the parameters given on the commandline are passed as paramters to `autogen.sh`.

- `configure` runs `configure` for each module. `--with-dune`*`module`* parameters are created for a set of known `DUNE` modules.
- `make` runs `make` for each module.
- `all` runs `autogen.sh`, `configure` and `make` for each module.

In order to build `DUNE` the first time you will need the `all` command. In pseudo code `all` does the following:

```
foreach ($module in $Modules) {
  foreach (command in {autogen,configure,make) {
    run $command in $module
  }
}
```

This differs from calling

```
dunecontrol autogen
dunecontrol configure
dunecontrol make
```

as it ensures that i.e. `dune-common` is fully built before `configure` is executed in `dune-grid`. Otherwise `configure` in `dune-grid` would complain that `libcommon.la` from `dune-common` is missing.

Further more you can add parameters to the commands; these parameters get passed on to the program being executed. Assuming you want to call `make clean` in all `DUNE` modules you can execute

```
dunecontrol make clean
```

opts files

You can also let `dunecontrol` read the command parameters from a file. For each command you can specify parameters. The parameters are stored in a varible called *`COMMAND`* `_FLAGS` with *`COMMAND`* written in capital letters.

Listing 3 (File examle.opts)

```
# use these options for configure if no options a provided on the cmdline
AUTOGEN_FLAGS="--ac=2.50␣--am=-1.8"
CONFIGURE_FLAGS="CXX=g++-3.4␣--prefix='/tmp/Hu␣Hu'"
MAKE_FLAGS=install
```

When you specify an opts file and command line paramters

```
dunecontrol --opts=some.opts configure --with-foo=bar
```

dunecontrol will ignore the parameters specified in the opts file and you will get a warning.

environment variables

You can further control the behavior of `dunecontrol` by certain environment variables.

- `DUNE_OPTS_FILE` specifies the opts file that should be read by dunecontrol. This variable will be overwritten by the `--opts=` option.
- `MAKE` tells `dunecontrol` which command to invoke for 'make'. This can be useful for exmaple, if you want to use *`gmake`* as a make dropin.
- `GREP` tells `dunecontrol` which command to invoke for 'grep'.

dune.module

The `dune.module` file is split into two parts. First we have the parameter section where you specify parameters describing the module. Then we have the command section where you can overload the default implementation of a command called via `dunecontrol`.

Listing 4 (File dune.module)

```
# paramters for dune control
Module: dune_grid
Depends: dune_common
Suggests: UG Alberta Alu3d

# overload the run_configure command
run_configure () {
  # lets extend the paramter list $PARAMS
  if test "x$HAVE_UG" == "xyes"; then
    PARAMS="$PARAMS \"--with-ug=$PATH_UG\""
  fi
  if test "x$HAVE_Alberta" == "xyes"; then
    PARAMS="$PARAMS \"--with-alberta=$PATH_Alberta\""
  fi
  if test "x$HAVE_Alu3d" == "xyes"; then
    PARAMS="$PARAMS \"--with-alberta=$PATH_Alu3d\""
  fi
  # call the default implementation
  run_default_configure
}
```

The parameter section will be parsed by `dunecontrol` will effect i.e. the order in which the modules are built. The parameters and their values are seperated by colon. Possible parameters are

- `Module` (*required*) is the name of the module. The name is of the form `[a-zA-Z0-9_]+`.
- `Depends` (*required*) takes a space seperated list of required modules. This module is not functional without these other modules.
- `Suggests` (*optional*) takes a space seperated list of optional modules. This module is functional without these other modules, but can offer further functionality if one or more of the suggested modules are found.

The command section lets you overload the default implementation provided by `dunecontrol`. For each command `dunecontrol` call the function `run_`*`command`*. The parameters from the commandline or the opts file are store in the variable `$PARAMS`. If you just want to create additional parameters you can add these to `$PARAMS` and then call the default implementation of the command via `run_default_`*`command`*.

3 Creating a new Dune module

`DUNE` modules are packages that offer a certain functionality that can be used by `DUNE` applications. Therefor `DUNE` modules offer libraries and/or header files.

In order to create new `DUNE` module, you have to provide

- a `dune.module` file
  Usually you will only need to specify the parameters `Module` and `Depends`.

- an `autogen.sh` script
  For most of the modules it should be sufficient to copy the `autogen.sh` from `dune-grid`.
- a `configure.ac` file
  Have look at the `configure.ac` in `dune-grid` for example. The most important part is the call to `DUNE_CHECK_ALL_M` which runs all checks needed for a `DUNE` module, plus the checks for the dependencies.

A `DUNE` module should comply with the following rules:

- Documentation is located under `doc/` and gets web-installed under `BASEDIR/doc/`.
- `automake` includes are located in `dune-common`. To use them, you will have to make a symbolic link to `dune-common/am/` (see 2.2.2). The symlink creation should be handled by the `autogen.sh` (see 2.4).
- The `am/` directory does not get included in the tarball.
- Header files that can be used by other `DUNE` modules should be accessible via `#include <dune/foo/bar.hh>`. In order to work with a freshly checkout version of your module you will usually need to create a local symbolic link `dune -> module-direcotry/`. This link gets created by the `DUNE_CHECK_ALL_M` command of your `configure.ac`. When running `make install` all header files should be installed into *prefix*`/include/dune/`.

4 Creating a new Dune application

A `DUNE` application does not differ a lot from a `DUNE` module. The only difference is that it does not offer functionality to other `DUNE` projects. This make somethings a little bit easier.

In order to create new `DUNE` module, you have to provide

- a `dune.module` file
  Usually you will only need to specify the parameters `Module` and `Depends`.
- an `autogen.sh` script
  For most of the application the `autogen.sh` following further below should be sufficient.
- a `configure.ac` file
  The `configure.ac` looks more less the same as for a `DUNE` module except that you call `DUNE_CHECK_ALL` instead of `DUNE_CHECK_ALL_M`.

Listing 5 (Example autogen.sh for a `DUNE` application)

```
#!/bin/sh

set -e

usage () {
    echo "Usage: ./autogen.sh [options]"
    echo "   --ac=, --acversion=VERSION    use a specific VERSION of autoconf"
    echo "   --am=, --amversion=VERSION    use a specific VERSION of automake"
    echo "   -h,    --help                 you already found this :-)"
}
```

```
for OPT in "$@"; do
    set +e
    # stolen from configure...
    # when no option is set, this returns an error code
    arg=`expr "x$OPT" : 'x[^=]*=\(.*\)'`
    set -e

    case "$OPT" in
    --ac=*|--acversion=*)
            if test "x$arg" == "x"; then
                usage;
                exit 1;
            fi
            ACVERSION=$arg
            ;;
    --am=*|--amversion=*)
            if test "x$arg" == "x"; then
                usage;
                exit 1;
            fi
            AMVERSION=$arg
            ;;
    -h|--help) usage ; exit 0 ;;
    *)
            if test -d "$OPT/m4"; then
              ACLOCAL_FLAGS="$ACLOCAL_FLAGS -I $(cd $OPT/m4; pwd)"
            fi
            if test -d "$OPT/am"; then
              am_dir="$OPT/am"
            fi
            ;;
    esac
done

if test x$1 = "x" ; then
  usage
  exit 0
fi

if test "x$ACLOCAL_FLAGS" = "x"; then
  echo dune-common/m4 not found. Please supply directory!
  usage
  exit 1
fi

if test -d m4 ; then
  ACLOCAL_FLAGS="$ACLOCAL_FLAGS -I m4"
fi

if test "x$AMVERS" != x ; then
  echo Warning: explicitly using automake version $AMVERS
  # binaries are called automake-$AMVERS
  AMVERS="-$AMVERS"
fi

aclocal$AMVERSION $ACLOCAL_FLAGS

libtoolize --automake --force

autoheader$ACVERSION

automake$AMVERSION --add-missing
```

```
autoconf$ACVERSION
```

5 Futher documentation

automake & Makefile.am

`http://www.gnu.org/software/automake/manual/`
The `automake` manual describes in detail how to write and maintain a `Makefile.am` and the usage of `automake`.

autoconf & configure.ac

`http://www.gnu.org/software/autoconf/manual/`
The `autoconf` manual covers the usage of `autoconf` and how to write `configure`.ac files (sometimes they are called `configure.in`).

Autoconf Macro Archive

`http://autoconf-archive.cryp.to/`
The Autoconf Macro Archive provides macros that can be integrated in your `configure.ac` in order to search for certain software. These macros are useful to many software writers using the autoconf tool, but too specific to be included into autoconf itself.

libtool

`http://www.gnu.org/software/libtool/manual.html`
The `libtool` manual offers further information on the usage of `libtool` package and gives a good overview of the different problems/aspects of creating portable libraries.

autobook

`http://sources.redhat.com/autobook/`
The autobook is a complete book describing the GNU toolchain (`autoconf`, `automake` and `libtool`). It contains many recipies on how to use the autotools. The book is available as an online version.